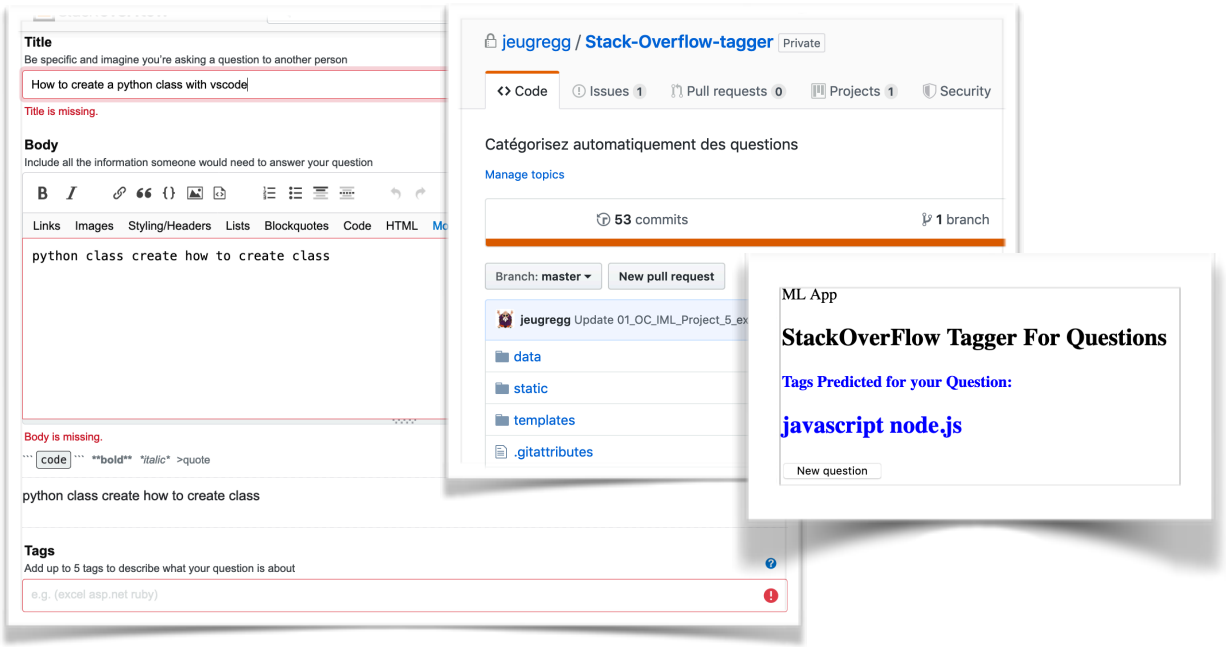
Grégory LANG
Openclassrooms
Parcours « Ingénieur Machine Learning »
7 novembre 2019

Catégorisez automatiquement des questions



Projet 5 IML

Résumé :

Ce rapport présente mon travail effectué pour le projet 5 Openclassrooms du parcours Ingénieur Machine Learning.

Ce projet consiste à proposer une API de prédiction de tags (mots-clés) en relation avec les questions posées par les utilisateurs sur le forum web StackOverFlow.

Ce document présente les différentes approches supervisées et non-supervisées, les traitements du texte tel que le «stemming», les modèles choisis et leur optimisation.
Aussi, une évaluation finale sera faite pour choisir le meilleur modèle.

Enfin, un modèle adapté est proposé pour tester une API de prédiction.
Cette API est opérationnelle pour le test : http://jeugregg.eu.pythonanywhere.com/

Sommaire

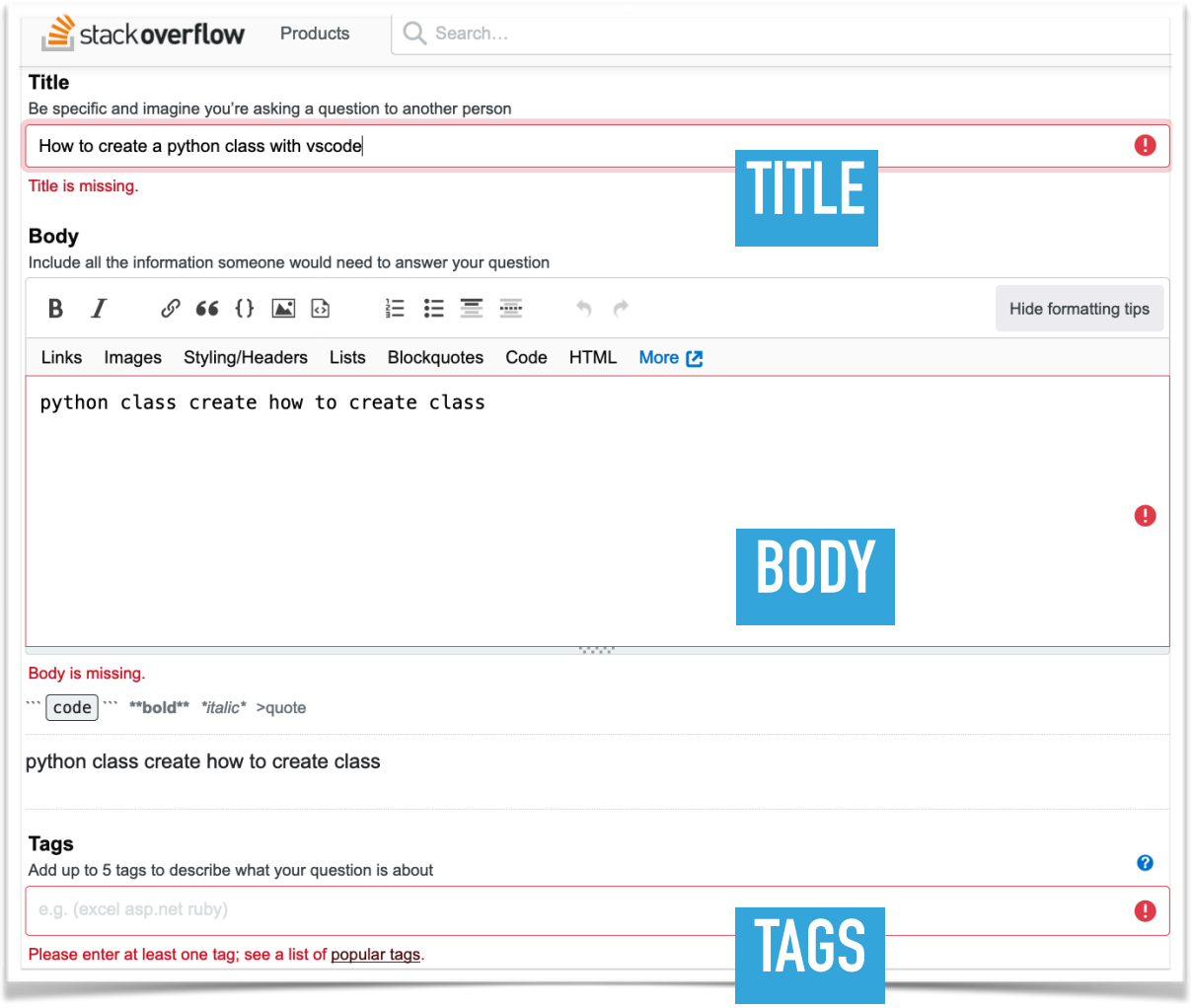
1. La problématique et son Interprétation

1.1. Contexte

Le site web StackOverFlow propose aux utilisateurs un forum de questions/ réponses où ils peuvent poser des questions techniques dans tous les domaines de la programmation informatique quelque soit le langage et le contexte.

Lors de l'ajout d'une question, 3 champs sont obligatoires :

- le titre de la question
- le contenu de la question
- et enfin au moins un tag (mot-clé)



(1) Formulaire web site StackOverFlow

1.2. Interprétation

Au lieu que l'utilisateur trouve seul des tags, on cherche à l'aider en lui proposant automatiquement des tags en relation avec sa question.

Le but est donc de trouver un moyen de détecter, à partir du texte saisi, les mots clés qui pourraient s'y rattacher.

1.3. Pistes envisagées

Deux principales approches sont possibles.

La première approche est celle qui utilise un modele non-supervisé, qui cherche les principaux sujets « topics », inconnus à l'avance, que peuvent traiter des sous-ensembles de questions déjà posées.
Ensuite, grâce aux questions déjà « taguées », on peut associer leurs tags avec les topics dominants des questions.

La deuxième approche est l'utilisation d'un modèle supervisée multi-label.
Cela implique d'avoir un nombre fini de tags prévisibles.

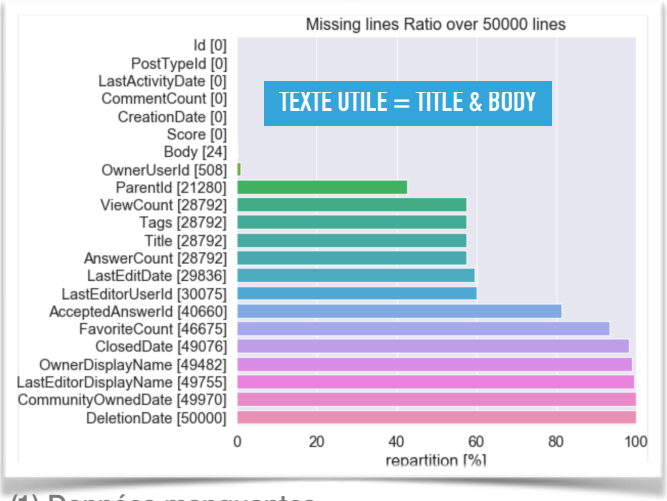
Ensuite, l'évaluation des deux approches permettra de faire un choix pour implémenter une API de test qui pourrait prédire des tags à partir d'un texte de question quelconque.

2. Cleaning effectué

Tout d'abord, on récupère les données depuis le site :
https://data.stackexchange.com/.
On ne peut récupérer que 50000 lignes à la fois.

On effectue le nettoyage suivant :
- éliminations des réponses.
- 24 Textes non-présents
- 41 Tags non-présents

reste : 21208 questions utiles

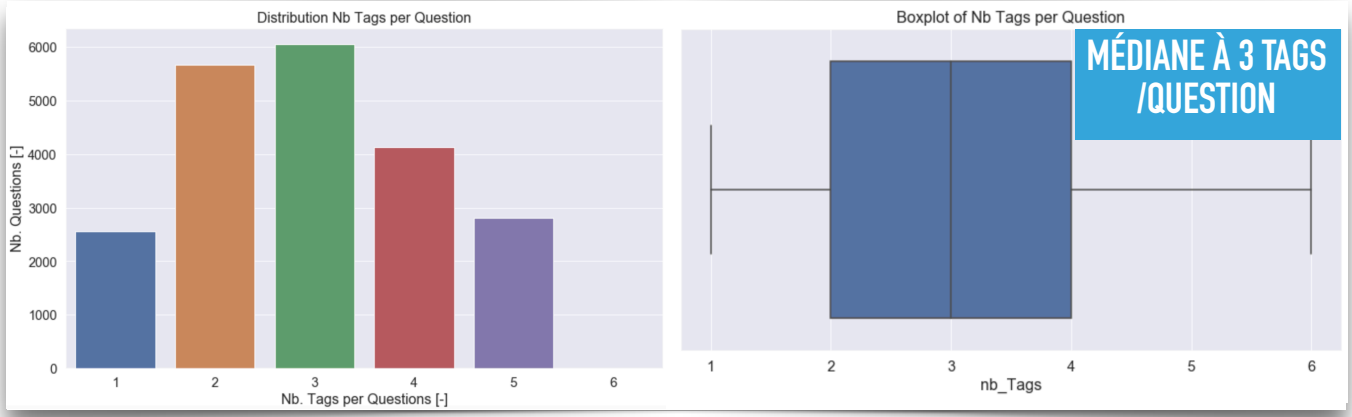


(1) Données manquantes

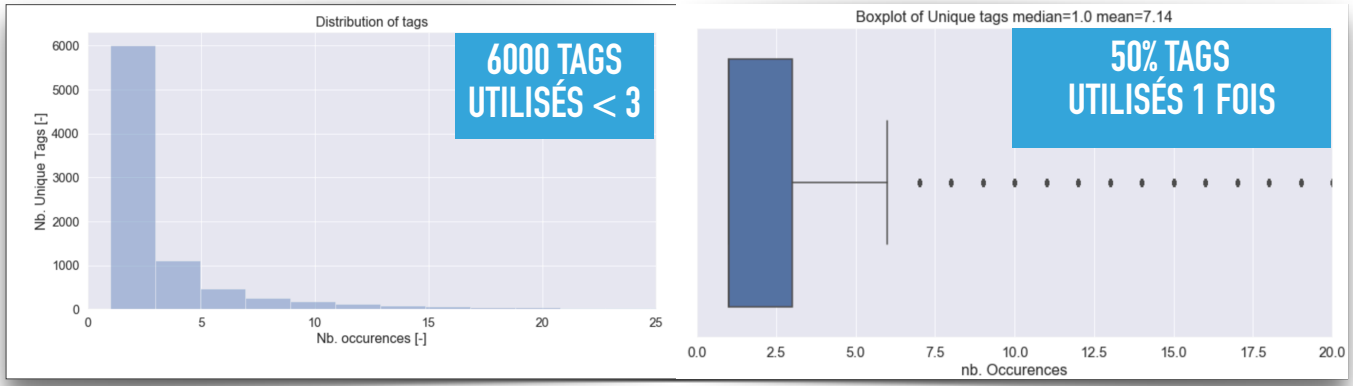
3. Exploration des tags

Il y a 8765 tags différents pour les 21208 questions.
50% des tags ne sont utilisés qu'une seule fois.
2/3 des tags ne sont utilisés que 2 fois au maximum.
Chaque question possède de 1 à 6 tags.
Le plus souvent, les questions ont entre 2 et 4 Tags.

8765 TAGS DIFFÉRENTS



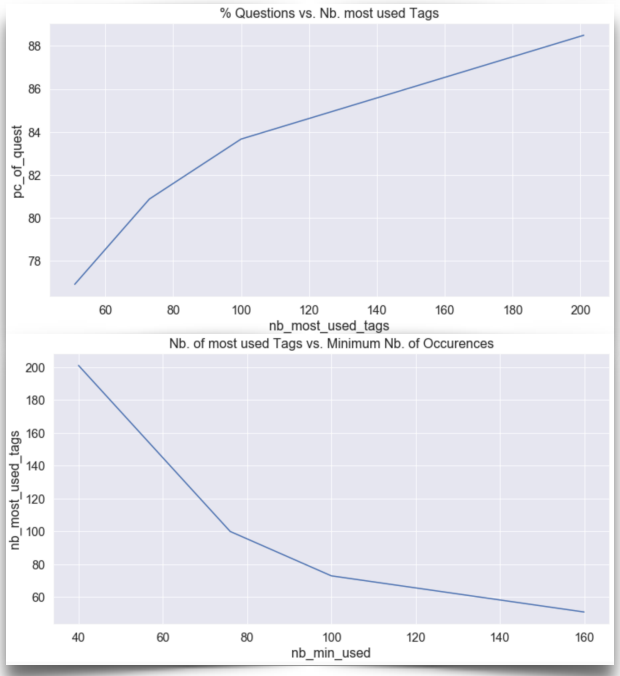
(2) Distribution nombre de tags par question



(3) Distribution des occurrences des différents tags

Pour les tags (73 tags) les plus utilisés (plus de 100 fois), bien qu'ils ne représentent que 1% des différents tags, ils sont tout de même inclus dans 81% des questions (cf. figure (4)).

L'évolution de l'utilisation des tags montre qu'au delà de 73 à 100 utilisations, le nombre des tags les plus utilisés diminue beaucoup plus lentement.



(4) Tags les plus utilisés

4. Préparation du texte

Il est nécessaire de préparer le texte pour créer des features pour les modèles.
Voici les étapes de cette préparation :

- Agrégation du « Title » et du « Body »

<p>2 Firebase Databases, and

- Mise en minuscule

<p>2 firebase databases, and

- Suppression caractères « espace» « tabulation » non désirés

<p>2 firebase databases, and

- Suppression tag HTML

2 firebase databases, and

- Suppression des nombres

firefase databases, and

- Suppression de la ponctuation

  remplacement par espace

firefase databases and

- Suppression des mots les plus utilisés

  création d'une liste de Stop Words =
  100 mots les plus utilisés +
  `nltk.corpus.stopwords.words('english'))`

firefase databases

- Création de tokens

  `nltk.word_tokenize(text)`

'firebase' 'databases'

- Extraction de la racine

  `EnglishStemmer()`

'firebas' 'databas'

4.1. Création des features

Chaque question est représentée par une liste de ses mots.
La création des features est basée sur le « term-frequency » de ses mots avec des seuils choisis pour éliminer les termes trop utilisés ou bien pas assez utilisés :
- limite mini d'utilisation d'un mot : 10 fois
- limite maxi de proportion de questions qui utilise un mot : 11%
- on ne garde que les 1000 mots les plus utilisés (**pour des raisons de mémoire**)

```
sklearn.feature_extraction.CountVectorizer(max_df=0.11,  min_df=10,
max_features=1000,…
```

5. Modèle non supervisé

Pour prédire les tags, le modèle non-supervisé doit passer par une étape identification des sujets communs (topics), inconnus à priori, et contenus dans les questions). La cible finale est la matrice les questions / tags : $M_1(Questions, Tags)$

Pour cela on utilise tout d'abord le modèle Latent Dirichlet Allocation à partir des features précédentes (TF). On obtient une distribution de topics par question.

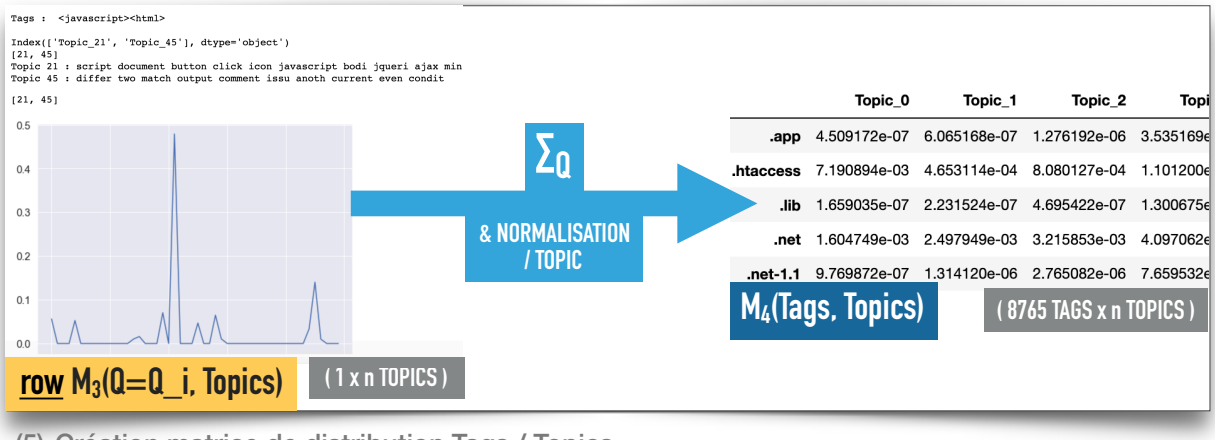
Matrice (Question,TF) : $M_2(Q, tf)$ donne distrib. topics : $M_3(Q, Topics) = LDA(M_2(Q, tf))$

Ensuite, connaissant les tags des questions d'entrainement, on peut trouver le lien entre les tags et les topics. Donc on crée une matrice (tags, topics) $M_4(Tags, Topics)$ tel que pour chaque ligne donc chaque tag, on ajoute les distributions de topics de toutes les questions contenant ce tag :

$$\underline{row}\ M_4(Tag=tag\_i, Topics) = \Sigma_Q\ \underline{row}\ M_3(Q=Q[Tag=tag\_i], Topics)$$

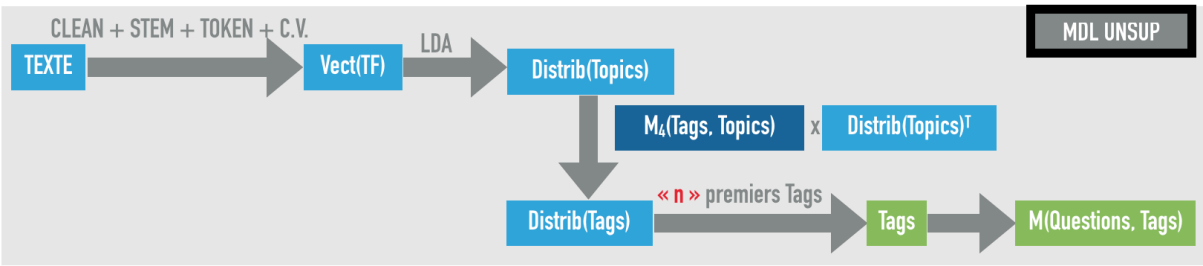
Puis, on normalise par topic pour avoir, par ligne, une distribution de topics par tag :

$$\underline{col}\ M_4(Tags, Topic=topic\_j) / \Sigma_{TAGS}\ \underline{col}\ M_4(Tags, Topic=topic\_j)$$



(5) Création matrice de distribution Tags / Topics

Enfin, en multipliant cette matrice par la distribution de topics d'une question, on obtient la probabilité de chaque tags pour cette question (cf. figure (6)).



(6) Diagramme de modélisation non supervisé de prédiction des Tags

5.1. Optimisation du modèle LDA

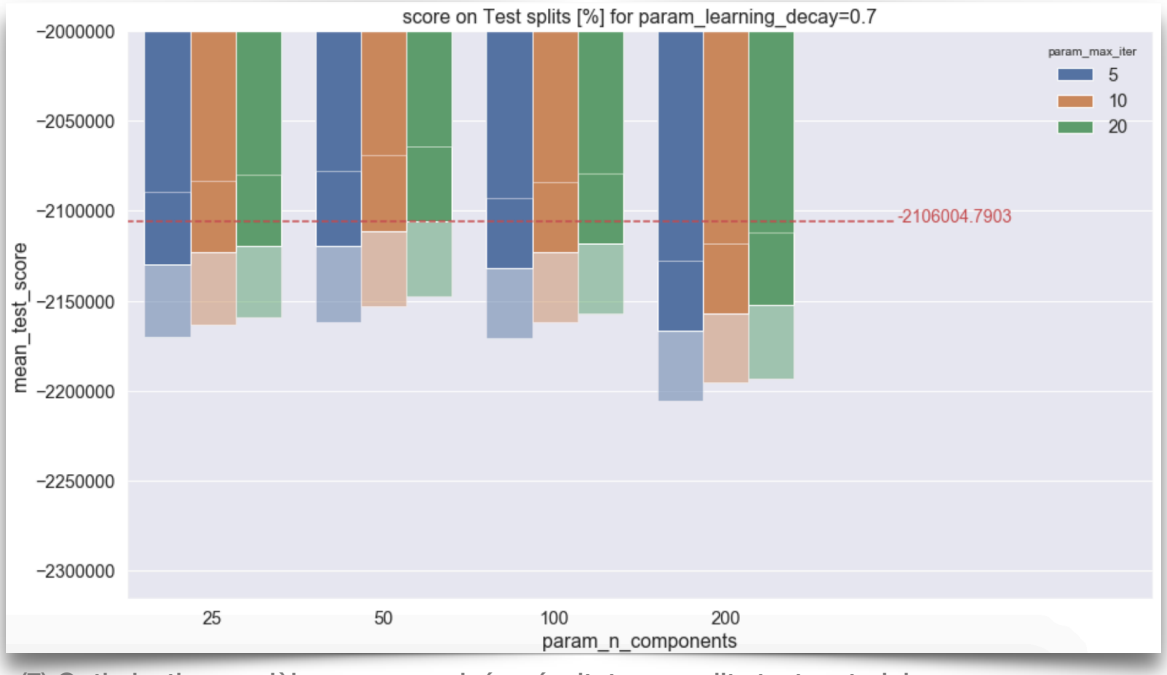
L'optimisation du modèle LDA (scikit-learn) est réalisée sur un jeu d'entrainement représentant 80% des 22000 questions grâce à une validation croisée sur 3 splits.

Voici les hyper-paramètres testés ainsi que le modèle :

```
search_params = {
      'n_components': [50],
      'learning_decay': [0.6, 0.7, 0.8],
      'max_iter': [5, 10, 20]}

LatentDirichletAllocation(learning_method='online',
      learning_offset=50,
      random_state=0,
      n_jobs=-1)
```

L'hyper-paramètre le plus impactant est le nombre de topics **n_components** à chercher.
Le meilleur score « log-likelihood » est obtenu avec **50 topics**.
On peut observer une légère amélioration du score avec **max_iter,** mais cela au détriment du temps de calcul qui devient beaucoup plus important.

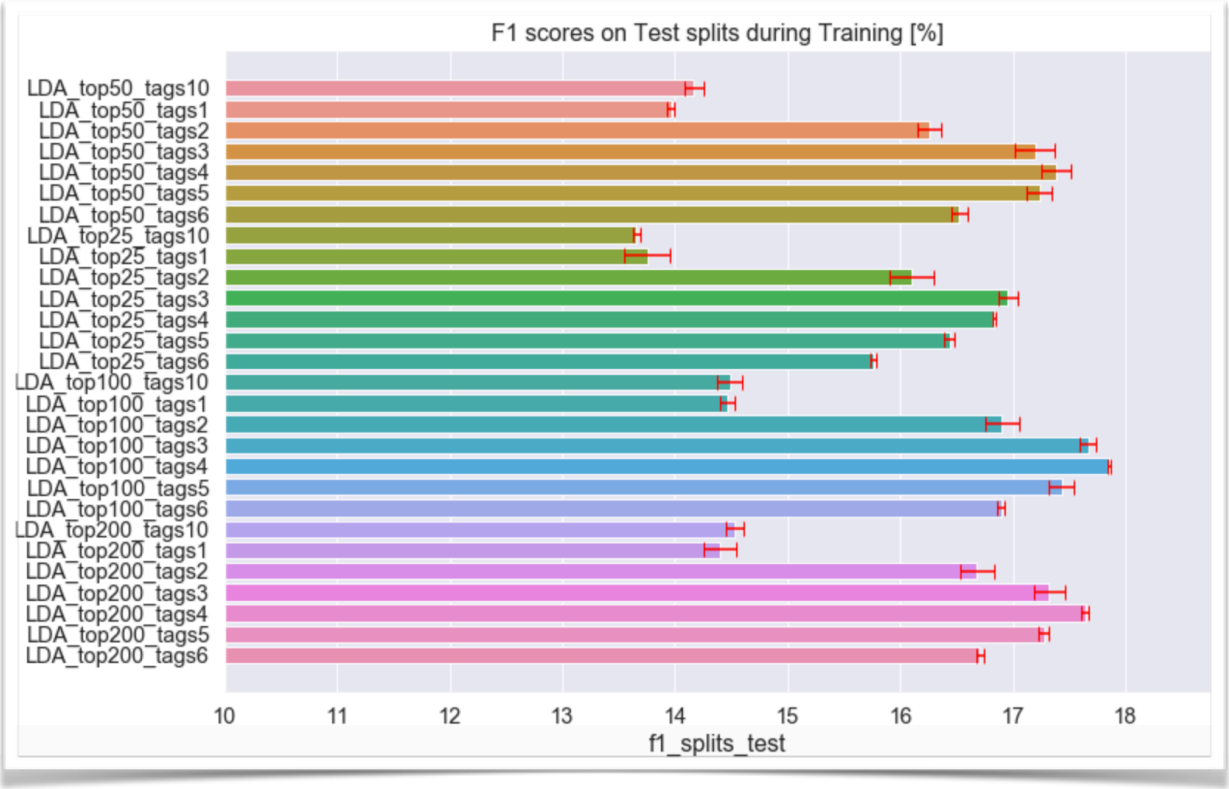


(7) Optimisation modèle non-supervisé : résultats sur splits test en training

5.2. Comparatif de prédiction sur « test splits »

Le meilleur F1-score sur les test splits sont les modèles prédisant 4 Tags.
Le paramètre le plus important est le **nombre de tags prédits** par question, c'est à dire les 4 premiers tags les plus probables.
Le nombre de topics **n_components** a un effet plutôt de 2ème ordre.

```
F1 = 2 * (precision * recall) / (precision + recall)
```



(8) Comparatif modèles LDA sur test splits avec intervalle de confiance (à 95%)

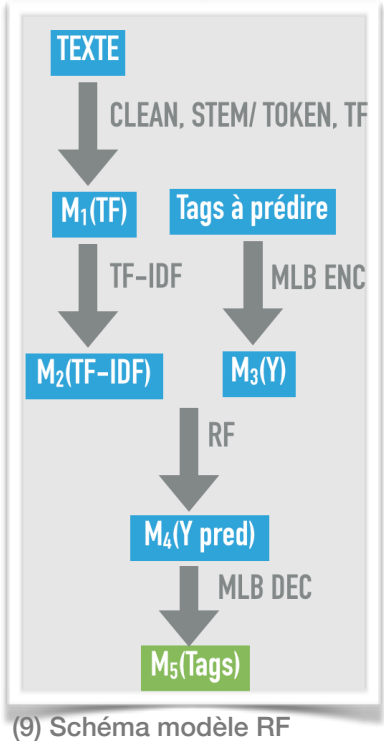
Comme le F1 score accorde autant d'importance à la sensibilité qu'a la précision, plus on prédit beaucoup de Tags, plus la précision et donc F1 s'écroule.

6. Modèle supervisé

Le modèle supervisé choisi est RandomForestClassifier. (cf. figure (9)). Cette fois-ci le modèle va prédire en mode multi-label. Les labels sont encodés pour l'apprentissage avec `MultiLabelBinarizer` de scikit-learn. A l'inverse, on les décode pour prédire.

Comme la mémoire est limitée et les données aussi, on ne peut prédire qu'un nombre maximum de Tags différents. Cela implique une réduction du jeu d'entrainement aux questions qui contiennent au moins un tag parmi les « n plus utilisés » (cf. figure (4)).

On utilise les 1000 Features TF transformées en TF-IDF de `TfidfVectorizer` de scikit-learn. Cela permet de prendre en compte la rareté d'un mot utilisé dans une question par rapport aux autres questions.

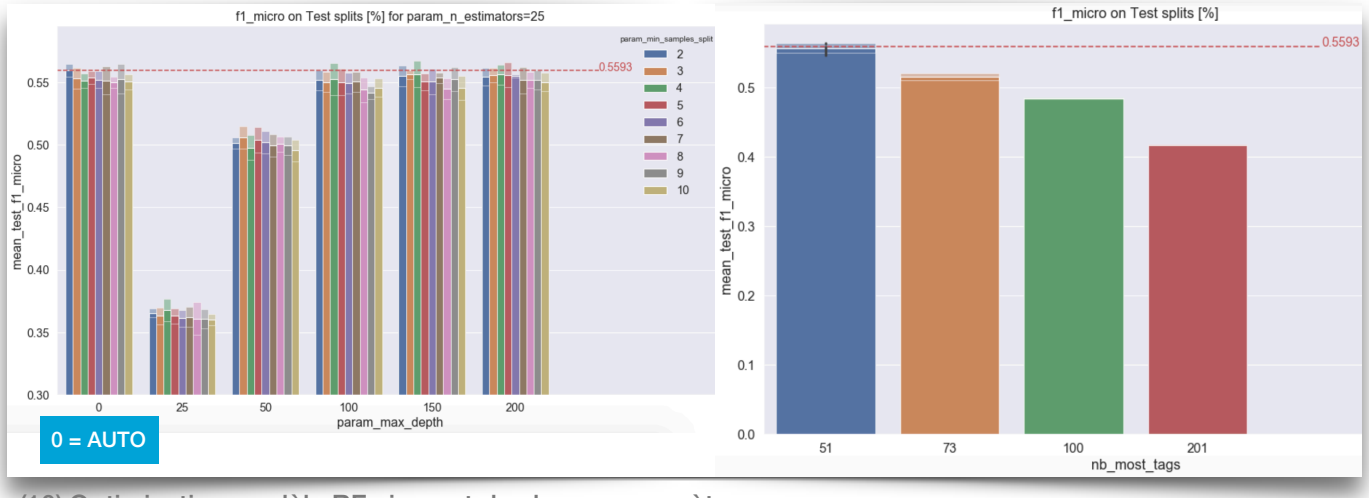


(9) Schéma modèle RF

6.1. Optimisation RF

L'optimisation des hyper-paramètres est faite avec une validation croisée à 3 splits sur 80% des questions maximum.
Les hyper-paramètres les plus importants sont :

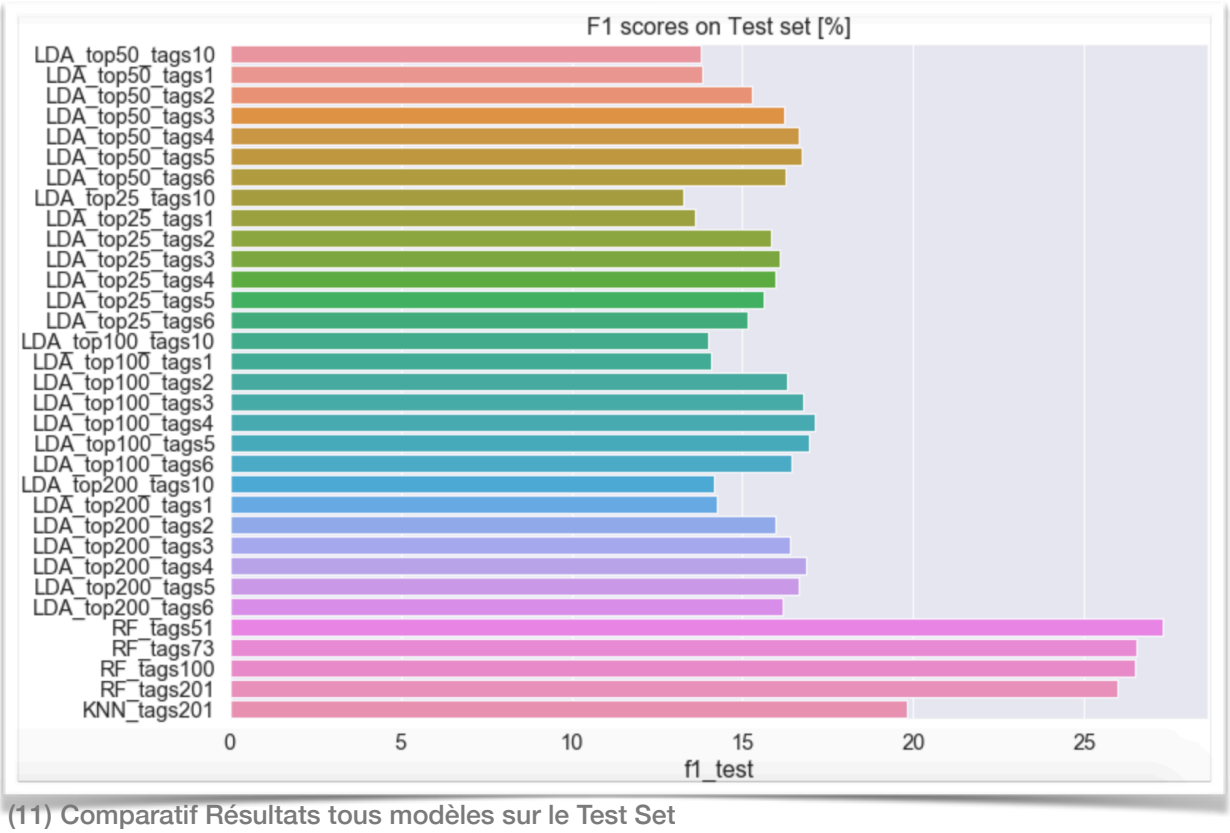
- **max_depth** (réduit F1 si trop faible),
- **nb tags à prédire** (réduit la sensibilité & F1 si trop important)
- Le meilleur modèle est celui limité à **51 Tags** différents (**nb_most_tags**).



(10) Optimisation modèle RF : impact des hyper-paramètres

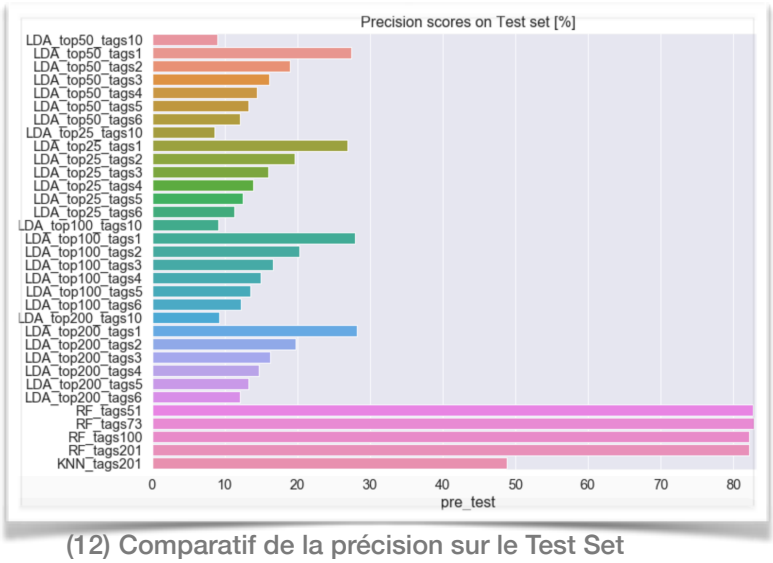
7. Evaluation globale

Les modèles supervisés ont un meilleur F1-score que les Non-Supervisés.
En réalité, ils sont beaucoup plus précis, bien qu'un peu moins sensible.
Leur défaut est simplement de parfois ne rien prédire. (cf. page suivante).



(11) Comparatif Résultats tous modèles sur le Test Set

Le modèle le plus généralisable est **RF 51 most_used_Tags** qui est aussi le meilleur en training.
On peut observer que les modèles non-supervisés ne sont pas très précis :



(12) Comparatif de la précision sur le Test Set

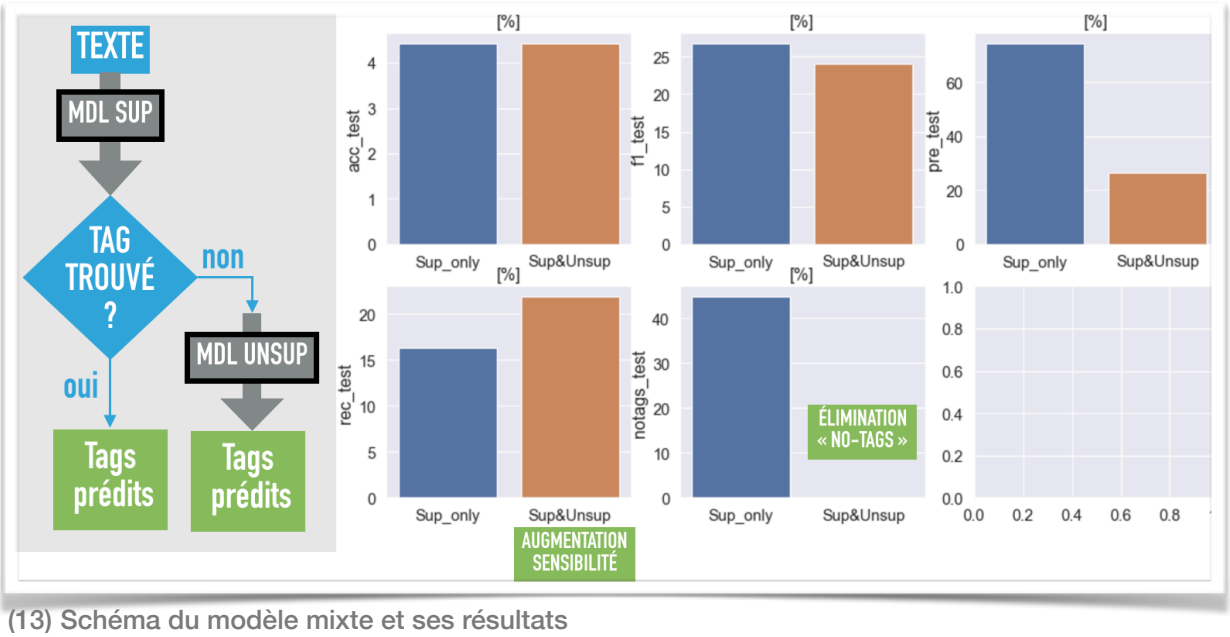
8. API

8.1. Modèle mixte

Le modèle mixte utilise d'abord un modèle Supervisé (**RF 51 Tags**).

Si on ne trouve pas de Tags, alors on utilise un modèle Non-Supervisé (**LDA 100 Topics 4 Tags**).

Cela améliore la sensibilité et élimine les « no-tags » (prédiction sans tags)



(13) Schéma du modèle mixte et ses résultats

8.2. API web app

L'intégration de API du modèle mixte a été réalisée sur:
http://jeugregg.eu.pythonanywhere.com/
Le code est suivi en version sur le repository GitHub suivant :
https://github.com/jeugregg/Stack-Overflow-tagger

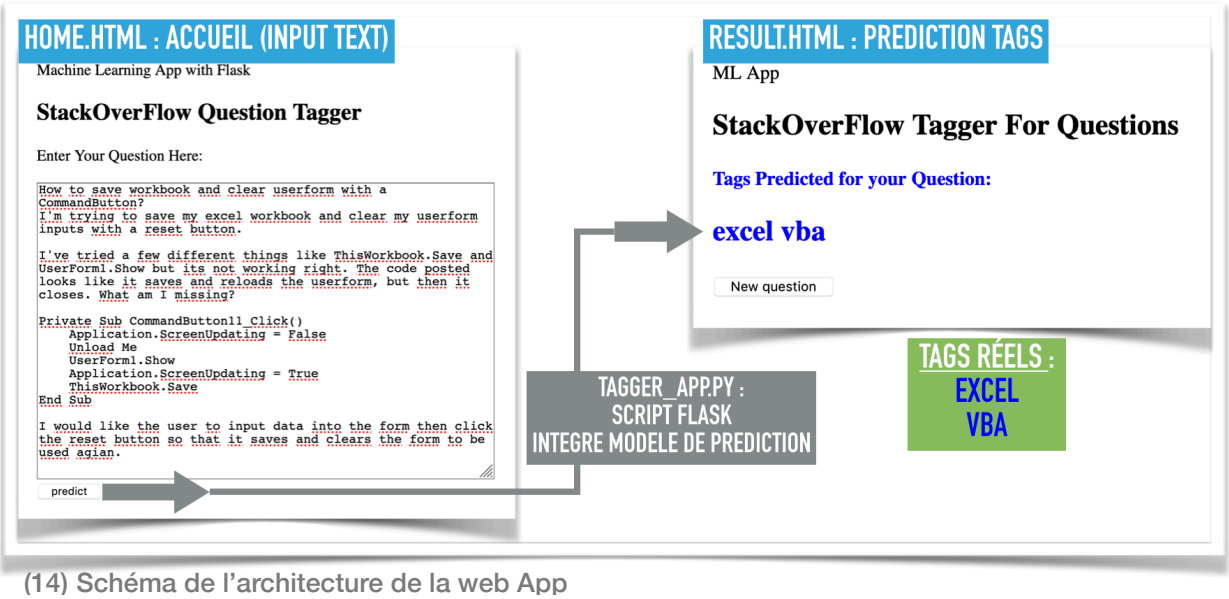
Plusieurs exports avec la librairie « pickle » des modèles et des données sont nécessaires :

Stop Words
CountVectorizer model RF
TF-IDF Transformer
Model RF
MultiLabelBinarizer (invert Transform)
CountVectorizer model LDA
Model LDA
TAGS/TOPICS DataFrame

L'architecture de la web app utilisant le framework Flask est simple :

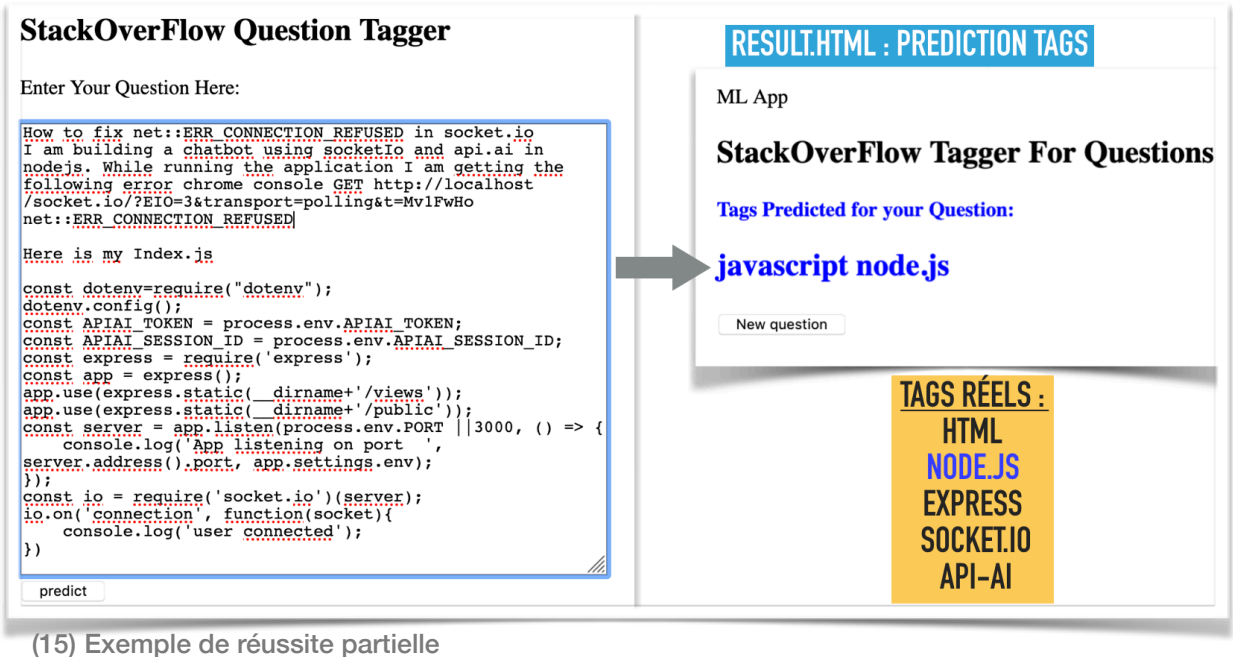
- **home.html** : formulaire d'entrée du texte de la question par l'utilisateur
- **result.html** : page de résultat
- **tagger_app.py** : « main function » en python pour gérer les pages et la prédiction

Voici un exemple de réussite de prédiction sur une question récente :



(14) Schéma de l'architecture de la web App

Un exemple de réussite partielle :



(15) Exemple de réussite partielle

9. Conclusions

Pour le problème de prédiction de Tags, les modèles non-supervisés ont une meilleure sensibilité, mais en se limitant à 4 Tags prédits pour garder de la précision.
Le meilleur modèle non-supervisé : **LDA avec 100 Topics 4 Tags prédits**.

Les modèles supervisés sont précis mais peu sensibles. Il arrive qu'ils ne prédisent pas de Tags du tout. Pourtant, le meilleur F1-score est obtenu en supervisé avec RandomForest. Attention, il faut limiter le nombre de différents Tags à prédire en supervisé car sinon on réduit la sensibilité.
Le meilleur score est obtenu par **RF 51 Tags** différents à prédire.
Les Modèles supervisés sont plus efficaces en temps de calcul : 10 fois plus rapide pour apprendre et 4 fois plus rapide pour prédire.

Pour une API, on propose un modèle mixte pour au moins prédire un Tag.

Une remarque : l'apprentissage est limité aux tags donnés par les utilisateurs. Mais les propositions de Tags alternatifs par le Modèle Non-Supervisé peuvent parfois être pertinentes.

10. Axes d'améliorations

Voici les principaux axes d'amélioration identifiés :

- Améliorer l'apprentissage avec plus de données en utilisant par exemple le Cloud Computing pour plus de puissance et plus de mémoire
- Apprentissage par morceau pour limiter l'utilisation de la mémoire
- Amélioration possible de la LDA avec plus d'itérations
- RandomForest ayant tendance à sur-apprendre : réduire cet effet en utilisant plus de données d'apprentissage
- Utiliser des modèles plus performants du type réseaux de Neurones pré-entrainés
- Intégration directe d'une API connectée au site StackOverFlow avec Stack Exchange API : https://api.stackexchange.com